맞춤된 COM 컴포넌트를 위한 효과적인 테스트 데이타 선정 기법과 적용사례

(An Effective Test Data Selection Technique for Customized COM Components and its Empirical Study)

윤 희 진 † 이 병 희 †† 김 은 희 ††† 최 병 주 ††††
(Hoijin Yoon) (Byunghee Lee) (Eunhee Kim) (Byoungju Choi)

요 약 컴포넌트 기반 개발에서 컴포넌트 사용자는 개발 목적에 맞추어 컴포넌트를 맞춤할 필요가 있다. 컴포넌트는 그 내부에 블랙박스특성을 갖는 부분과 화이트박스특성을 갖는 부분이 공존하고, 맞춤으로 인해 화이트박스 부분이 변형되며, 이때 발생하는 오류는 블랙박스 부분과 화이트박스 부분의 상호작용을 통해 나타나게 된다. 블랙박스 부분과 화이트박스 부분 사이의 통합 테스트인 맞춤 테스트를 위해서 새로운 테스트 기법이 요구된다. 또한 테스트 기법이 비용 절감 효과를 노리는 컴포넌트 기반 개발에 사용되기 위해서는 효율적인 테스트 데이타의 선정이 요구된다. 따라서 본 논문에서는 컴포넌트 아키텍쳐로 COM(Component Object Model)을 대상 컴포넌트로 선정하고, 다양한 COM 컴포넌트들의 분석을 통하여 효율적인 테스트 데이타를 선정하는 맞춤 테스트 기법을 제안한다.

본 논문에서는 제안하는 기법이 선정하는 테스트 데이타가 오류 감지 능력에 있어서 효과적이라는것을 실험을 통해 평가한다. 또한 본 기법을 실제 대규모 컴포넌트 기반 시스템인 샤모아에 적용하는 예제를 수행함으로써, 본 기법이 컴포넌트 기반 시스템의 일원으로서 실제 동작하는 COM 컴포넌트의 맞춤을 테스트할 수 있음을 보인다.

키워드 : 컴포넌트 맞춤 테스트, 효율성, 컴포넌트 기반 소프트웨어 개발, COM

Abstract Component users must customize components they obtain from providers, in order to fit them to their own purposes. Normally, a component consists of black-box parts and white-box parts. Component users customize a component by modifying white-box parts of a component, and the customization faults appear through the interaction between black-box parts and white-box parts. Customization testing could be an integration testing of these two parts of a component. Also, customization testing in CBSD should select effective test data to reduce the testing cost, since CBSD aims to reduce the development cost. Therefore, this paper proposes a customization testing technique based on COM architecture through analyzing many COM components, and the technique selects effective test data.

This paper evaluates the effectiveness of the test data selected by the proposed technique through an empirical study. It applies the technique to a large-scale component-based system, Chamois, and it shows that the technique enables us to test customized COM components that run in a real component-based system

Key words : Component Customization Test, Effectiveness, Component-based Software Development, COM

· 본 연구는 과학재단 목적기초연구(R04-2003-000-10139-0) 지원으로 연구되었음

† 비 회 원 : Georgia Institute of Technology College
981COG04@ewha.ac.kr
†† 비 회 원 : (주)키움닷컴증권 투자정보시스템팀 근무
bhlee@ewha.ac.kr
††† 정 회 원 : 삼성전자 기술총괄 CTO 전략실
ehkim@ewha.ac.kr
†††† 종신회원 : 이화여자대학교 컴퓨터학과 교수
bjchoi@ewha.ac.kr

논문접수 : 2003년 2월 4일
심사완료 : 2004년 3월 26일

1. 서 론

컴포넌트 기반 소프트웨어 개발은 컴포넌트 맞춤 테스트(Component Customization Test), 컴포넌트 조립 테스트(Component Composition Test), 컴포넌트 시스템 테스트(Component System Test), 컴포넌트 자격부여 테스트(Component Qualification Test) 등의 다양한 수준의 테스트를 요구한다[1]. 이 가운데 맞춤 테스트는 컴포넌트 사용자가 컴포넌트 기반 개발에서 반드시 수행해야 하는 테스트 수준이다. 컴포넌트 사용자는 컴포넌트 제공자로부터 얻은 컴포넌트를 자신의 도메인 특성에 맞도록 맞춤할 필요가 있다. 이때 컴포넌트는 그 핵심 부분의 코드를 제공하지 않고, 단지 인터페이스라는 일부분만을 공개함으로써, 이 부분을 통하여 컴포넌트를 맞춤하도록 한다. 맞춤 테스트 대상이 되는 컴포넌트는 블랙박스 부분과 화이트 박스 부분이 공존하므로, 기존의 블랙박스 테스트나 화이트 박스 테스트만을 가지고는 테스트하는데 무리가 있다. 따라서 이러한 컴포넌트의 특성을 고려한 새로운 맞춤 테스트 기법이 요구되며, 본 연구진은 이미 맞춤 테스트에 대한 다양한 연구를 진행해 오고 있다.

우선 본 연구진은 공개하지 않는 컴포넌트의 핵심기능 부분과 공개하는 컴포넌트의 인터페이스 부분을 각각 블랙박스 클래스와 화이트박스 클래스로 정의하고, 컴포넌트 맞춤을 통해 변형된 화이트박스 클래스에 오류를 삽입하여 맞춤 오류를 테스트하는 테스트 데이타를 선정하는 기법을 제안하였다[2]. 또한 이 기법이 컴포넌트 맞춤에 효율적인 테스트 데이타를 선정할 수 있도록 함을 엔터프라이즈 자바빈즈에 적용한 사례와 실험을 통하여 나타내었다[2]. 이 실험에는 제안한 기법을 자동으로 수행하는 엔터프라이즈 자바빈즈 맞춤 테스트 도구를 직접 구현하여 사용하였다[3].

본 논문에서는 COM(Component Object Model)의 사양을 분석하여 COM 컴포넌트에 적용할 수 있는 기법을 개발하고, 샤모아[4]라는 대규모 컴포넌트 기반 시스템을 대상으로 기법을 적용한 사례를 보인다. 샤모아 컴포넌트 시스템은 여러 상용 제품 및 자체 개발한 컴포넌트들을 IKEA(Integrated Knowledge Engineering Architecture)구조 하에 컴포넌트 개발 개념에 따라 개발한 시스템이다. 샤모아 컴포넌트 시스템은 COM 기반의 컴포넌트들로 이루어져 있으므로, COM 컴포넌트 아키텍쳐를 분석하여 본 논문에서 제안하는 COM 컴포넌트를 위한 기법을 적용한다. 샤모아 컴포넌트 시스템의 데이타 품질 측정 도구인 DAQUM 컴포넌트의 맞춤 테스트에 적용하는 예제를 보인다. 나아가 테스트 기법이 비용 절감 효과를 노리는 컴포넌트 기반 개발에 사용되기 위해서는 효율적인 테스트 데이타의 선정이 요구되므로, 본 기법이 선정하는 테스트 데이타의 오류 발견 가능성에 대한 효율성을 실험을 통하여 평가하고 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 기술하고, 3장에서는 COM 컴포넌트 맞춤 테스트 기법을 기술한다. 4장에서는 샤모아 컴포넌트 시스템 내에 적용한 사례연구를 보이며, 5장에서는 실험과정과 결과를 기술하고 분석한다. 마지막으로 6장에서 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 컴포넌트 맞춤 테스트

컴포넌트 사용자는, 컴포넌트의 소스코드를 사용할 수 없기 때문에 컴포넌트 사용자 입장에서 수행하는 테스팅에 제약이 있다. 컴포넌트 사용자는 컴포넌트 맞춤, 즉 컴포넌트를 특정 도메인 요구사항에 맞추는 작업을 반복적으로 수행함으로써, 컴포넌트 기반 소프트웨어를 개발한다. 따라서 컴포넌트 맞춤으로 변형된 컴포넌트를 테스트하기 위한 컴포넌트 맞춤 테스트 기법이 필요하다.

이미 컴포넌트의 일반적인 특성을 기반으로 컴포넌트 맞춤 테스트 기법[2]이 제안되었다. 컴포넌트 맞춤 테스트는 컴포넌트의 맞춤으로 변형된 인터페이스가 공개되지 않은 핵심기능 부분에 영향을 미치면서 컴포넌트 행위에 일으키는 오류를 테스트하는 컴포넌트의 핵심기능 부분과 인터페이스의 통합 테스트이다. 따라서 인터페이스를 구성하는 구성요소들 가운데, 컴포넌트의 핵심 기능 부분과 밀접한 관계를 맺는 부분을 오류 삽입 대상으로 선정하여 오류 삽입 기법과 뮤테이션 테스트 기법을 적용하였다[2,3]. 본 논문에서는 실제 샤모아 컴포넌트 시스템에 적용할 수 있도록 COM 컴포넌트 맞춤 테스트 기법을 개발하고, 이를 샤모아에 적용한다.

컴포넌트에 적용할 수 있는 다른 기법으로는 인터페이스 뮤테이션을 들 수 있다. 인터페이스 뮤테이션[5,6]은 본 기법과 유사하게 전체 문장을 변형시키지 않고, 단지 다른 모듈을 호출하는 문장만을 변형시키는 방법을 통해, 두 모듈 사이의 통합 테스트를 수행하는 기법이다. 이는 호출하는 문장의 매개변수, 호출되는 모듈의 반환변수, 그리고 전역변수를 변형 대상으로 오류를 임의로 심어 뮤턴트를 생성한다. 그러나 본 논문에서는 인터페이스 뮤테이션과 같이 메소드 호출 부분을 중심으로 오류를 심지 않고, COM 컴포넌트들의 맞춤 패턴을 추출하고, 각 맞춤 패턴에 따라 오류를 삽입할 특정 부분을 찾아서 그곳에만 오류를 심는다. 이러한 차이점이 테스트 데이타의 효율성 면에서 어떻게 나타나는지를 4장의 실험을 통해 보인다.

2.2 COM

COM은 마이크로소프트사가 제안한 프로그램의 컴포넌트 객체들을 개발하고 지원하기 위한 하부 기반구조이다. 강력한 프로그램의 이식성과 컴포넌트 개발의 유연성을 제공하고, 재컴파일이 필요없는 컴포넌트의 업그레이드, 효율적인 소프트웨어의 모듈화, 플랫폼의 독립성등의 장점을 가진다[7]. COM은 마이크로소프트사의 Visual C++, Visul Basic, .NET등 다양한 언어로 구현 가능하며, 이들 간의 확장성이 뛰어나 많은 개발자들이 COM을 기반으로 컴포넌트를 개발하고 있다.

COM 컴포넌트는 컴포넌트의 클래스들로 구성된 객체와 클라이언트와 통신하기 위한 하나 이상의 인터페이스로 구성된다. 이들 COM 컴포넌트는 마이크로소프트사가 표준화한 규정에 맞게 작성된 DLL혹은 EXE형식으로 배포된다. 객체는 COM 컴포넌트의 주요 기능을 수행하는 클래스들로 구현되어 소스코드를 제공하지 않는 블랙박스이고, 인터페이스는 COM의 객체가 자신의 기능을 노출시키는 것으로 맞춤이 가능한 화이트박스 영역이다.

3. COM 컴포넌트 맞춤 테스트 기법

컴포넌트 맞춤 테스트는 컴포넌트의 인터페이스 부분과 컴포넌트의 실제 구현 코드를 갖는 부분의 통합테스트라고 할 수 있다. 이들 두 부분을 각각 블랙박스 클래스와 화이트 박스 클래스로 정의하여 컴포넌트의 특성을 표현하였다[2,3]. 또한 맞춤 테스트 데이타를 선정하기 위한 오류를 삽입하는 부분을 특별히 오류 삽입 대상으로 정의하고 오류를 삽입하는 연산자를 오류 삽입 연산자로 정의하였다. 본 논문에서는 이전 논문에서 정의해 놓은 개념들을 사용하여 COM 컴포넌트를 위한 맞춤 테스트 기법을 제안한다. 본 장에서는 먼저 본 논문에서 사용할 개념들에 대한 정의를 하고, 이들 개념들 기반으로 COM 컴포넌트를 분석하여 COM 컴포넌트에 실제 적용할 수 있는 수준의 테스트 방안을 제안한다.

3.1 개념 정의

정의 1. 블랙박스 클래스(B)

블랙박스 클래스란 컴포넌트의 핵심기능을 구현한 부분으로서, 컴포넌트 사용자에게 코드가 공개되지 않는 부분이다. 결국 이 부분은 컴포넌트 사용자가 수정할 수 없는 부분이다. 이를 B로 표현한다.

정의 2. 화이트박스 클래스(W)

화이트박스 클래스란 컴포넌트의 인터페이스 부분으로서 컴포넌트 사용자에게 공개된 부분이다. 컴포넌트 사용자는 이 부분을 이용하여 컴포넌트를 맞춤할 수 있다. 이를 W로 표현한다. 특히 맞춤된 화이트박스 클래스를 cW로, 그리고 오류가 삽입된 화이트박스 클래스를 fW로 표시한다.

정의 3. 컴포넌트(BW)

컴포넌트는 B와 W가 결합된 단위로서 BW로 표현한다. 컴포넌트 맞춤으로 W가 변형된 BW를 cBW라고 하고, 오류가 삽입된 cBW를 fBW로 표현한다.

맞춤된 컴포넌트를 테스트하는 테스트 데이타를 구하기 위하여 임의의 오류를 삽입한 fBW를 생성한다. 이때 오류를 cW전체에 삽입하거나, 또는 cW내에서 맞춤으로 인해 수정된 부분에 삽입하는 것이 아니라, W와 B의 상호작용을 분석하여, B에 직접 영향을 주는 부분에 오류를 삽입한다. 이렇게 함으로써, 선정된 테스트 데이타가 맞춤으로 인한 오류를 감지하는데 좋은 효율성을 갖게 된다. 본 논문에서는 오류가 삽입 되는 부분을 오류 삽입 대상으로, 그리고 오류를 삽입하는 방법인 연산자를 오류 삽입 연산자로 정의한다.

정의 4. 오류 삽입 대상(FIT)

cW에서 오류를 삽입하는 특정한 부분을 오류 삽입 대상으로 정의하고, 이를 FIT로 표시한다. FIT는 선정된 테스트 데이타에 효율성을 제공한다.

정의 5. 오류 삽입 연산자 (FIO)

오류 삽입 연산자는 cBW가 문법적 오류를 일으키지 않는 범위내에서 오류 삽입 대상에 어떤 오류를 어떻게 심을것인지를 나타내는 연산자이다. 이는 FIO로 표시한다.

위의 정의들에 더하여 한가지 더 의미있는 방안이 추가된다. 컴포넌트 맞춤 패턴이 그것이다. FIT와 FIO를 적용하기에 앞서, 우선 cBW가 어떤 맞춤 패턴에 의해서 맞춤 되었는지를 살피고, 맞춤 패턴에 따라 달리 정의된 FIT와 FIO들을 찾아 그들만을 cBW에 적용하여 테스트 데이타를 선정한다. 맞춤 패턴은 맞춤으로 인한 오류에 민감한 테스트 데이타를 선정하는데 도움이 된다. 이전 논문들에서 이미 컴포넌트 맞춤 패턴을 분류하였다. 맞춤 패턴은 두 개의 관점을 가지며, 하나는 맞춤의 구문적 관점이고 또 다른 하나는 맞춤의 의미적 관점이다. 이들을 각각 *Pattern.syn*과 *Pattern.sem*으로 표시된다. 다음과 같이 *Pattern.syn*에는 두 개의 맞춤 패턴이 있고, *Pattern.sem*에는 세 개의 맞춤 패턴이 있다[2,3].

- $Pattern.syn_1$: W에 맞춤 코드를 직접 삽입하는 맞춤
- $Pattern.syn_2$: 맞춤 코드를 갖는 새로운 클래스를 하나 만들고, 이를 기존의 W에 연관관계로 연결하는 맞춤
- $Pattern.syn_3$: 맞춤 코드를 갖는 새로운 클래스를 하나 만들고, 이를 기존의 W에 상속관계로 연결하는 맞춤
- $Pattern.sem_1$: 컴포넌트의 특성들을 수정하기 위하

여 W를 변형하는 맞춤

- $Pattern.sem_2$: 컴포넌트의 기능을 수정하거나 추가하기 위하여 W를 변형하는 맞춤

그림 1은 위의 개념들을 컴포넌트 맞춤 테스트가 어떻게 이용하는지를 보여준다. B와 cW는 cBW로 통합되어, 하나의 COM 컴포넌트가 된다. 비록 B와 cW 각각은 오류가 없는 단위일지라도, 이들이 통합되었을 때 예기치 못한 치명적인 오류가 발생할 수 있다. 컴포넌트 맞춤 테스트 기법[2]은, 그림 1(a)의 '컴포넌트 맞춤'에서 보듯이 컴포넌트 맞춤을 통해 변형된 컴포넌트, 즉 cBW의 오류를 테스트하기 위해, 그림 1(b)의 '오류 삽입'을 통해 cBW에 임의로 오류를 삽입하여 생성한 fBW와 cBW를 차별화하는 테스트 데이타를 선정한다. 선정된 테스트데이타를 cBW에 적용시켜 그 결과가 예상 결과와 다를 경우, cBW에 오류가 있다고 판단한다.

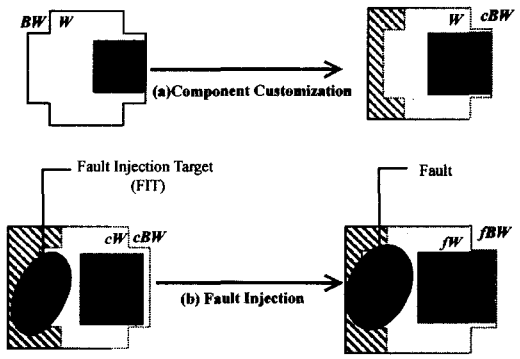


그림 1 컴포넌트 맞춤과 오류 삽입

컴포넌트 맞춤 테스트의 테스트 데이타는 일련의 메소드들이다. 맞춤 테스트 기법은 이들 일련의 메소드들을 cBW와 fBW를 이용하여 선정한다. 즉, cBW와 fBW를 차별화할 수 있는 일련의 메소드들을 테스트 데이타로 선정한다. 이는 정의 6으로 표현할 수 있다.

정의 6. 맞춤 테스트 데이타 선정

TC는 테스트 데이타의 집합이다.

$TC=\{x|cBW(x) \neq fBW(x)$, x는 메소드 시퀀스이다.$\}$

정의 6은 뮤테이션 테스트 데이타와 유사하다. 그러나 이는 뮤테이션 테스트처럼 cBW의 모든 구성요소들을 모두 뮤테이트시키는 것이 아니라, 맞춤 패턴에 따라 해당되는 FIT에만 FIO를 이용하여 오류를 심음으로써 fBW를 생성한다. 이는 컴포넌트 맞춤 테스트에 보다 효율적인 테스트 데이타를 선정할 수 있도록 한다. 테스트 데이타의 효율성은 4장에서 보여진다.

3.2 COM(Component Object Model) 컴포넌트의 맞춤 패턴

COM 컴포넌트는 객체와 인터페이스로 이루어진다. 객체는 컴포넌트의 주요 기능을 구현하는 클래스들로 구성되며, 그 소스코드를 공개하지 않으므로 블랙박스 클래스인 B가 될 수 있다. 인터페이스는 외부에서 접근 가능하도록 만들어져 있으므로 화이트박스 클래스인 W로 정의한다.

컴포넌트 맞춤을 통해, BW를 cBW로 만드는 데에는 몇 가지 패턴이 있다. 이 패턴에 따라 cBW에 오류를 삽입하는 대상과 내용이 달라진다. 따라서 본 장에서는 COM 컴포넌트의 맞춤 패턴을 정의하고, 그들에 따라 COM 컴포넌트의 오류 삽입 대상을 선정한다.

COM 컴포넌트의 맞춤 패턴은 표 1과 같다.

표 1 COM 컴포넌트의 맞춤패턴

맞춤 패턴	설명
패턴1	인터페이스를 변형하여 컴포넌트의 특성 값들을 설정하는 맞춤
패턴2	새로운 인터페이스를 생성하여 컴포넌트의 특성 값들을 설정하는 맞춤
패턴3	인터페이스를 변형하여 새로운 기능을 컴포넌트에 추가하는 맞춤
패턴4	새로운 인터페이스를 생성하여 새로운 기능을 컴포넌트에 추가하는 맞춤
패턴5	인터페이스에 새로운 시퀀스를 추가하여 컴포넌트의 기능을 변형하는 맞춤
패턴6	새로운 인터페이스에 새로운 시퀀스를 추가하여 컴포넌트의 기능을 변형하는 맞춤

COM 컴포넌트의 맞춤 패턴들을 각각 앞서 정의한 $Pattern.syn$과 $Pattern.sem$에 표 2와 같이 각각 해당된다.

표 2 COM 맞춤 패턴과 $Pattern.syn$, $Pattern.sem$

Pattern.sem \ Pattern.syn	Pattern.syn1	Pattern.syn2	Pattern.syn3
Pattern.sem1	패턴1	패턴2	
Pattern.sem2	패턴3, 패턴5	패턴4, 패턴6	

COM 맞춤 패턴이 정의되었으면, 이들을 기반으로 각 패턴에서의 FIT와 FIO를 정의 4와 정의 5에 따라 선정한다. 앞서 언급했듯이, FIT는 맞춤을 통해 변형된 부분이 아니고, B와 직접 관계를 맺는 W의 특정 부분이다. 따라서 COM 컴포넌트의 FIT와 FIO를 선정하기 위해서는 다수의 실제 COM 컴포넌트들을 대상으로 맞춤을 수행하고 각 맞춤에서 W와 B의 상호관계를 분석하여야만 한다. 본 논문에서는 비쥬얼베이직으로 코드된 COM 컴포넌트들을 대상으로 분석을 하였다. 각 패턴에서의 FIT와 FIO는 표 3과 같다.

표 3 COM 컴포넌트의 FIT와 FIO

맞춤 패턴	*FIT*s	*FIO*s
패턴1	*Dim*: "Dim"으로 선언된 특성	*MDim*: "Dim"으로 선언된 특성의 값을 변형
	Form: "Form"으로 선언된 특성	*MForm*: "Form"으로 선언된 특성의 값을 변형
패턴2	*DimN*: 새로운 인터페이스에서 "Dim"으로 선언된 특성	*MDimN*: 새로운 인터페이스에서 "Dim"으로 선언된 특성의 값을 변형
	FormN: 새로운 인터페이스에서 "Form"으로 선언된 특성	*MFormN*: 새로운 인터페이스에서 "Form"으로 선언된 특성의 값을 변형
패턴3	*PublicSub*: "Public Sub"로 선언된 기능	*RSub*: "Public Sub"로 선언된 기능을 새로운 컴포넌트로 대치
		CSub: 새로운 기능 추가
	PublicFunction: "Public Function"으로 선언된 기능	*RFnt*: 기능을 새로운 컴포넌트로 대치
		MPrm: 새로운 기능의 매개변수의 값 변형
패턴4	*PublicSubN*: 새로운 인터페이스에서 "Public Sub"로 선언된 기능	*RSubN*: 새로운 인터페이스에서 "Public Sub"로 선언된 기능을 새로운 컴포넌트로 대치
		CSubN:새로운 인터페이스에 기능 추가
	PublicFunctionN: 새로운 인터페이스에서 "Public Function"으로 선언된 기능	*RFntN*: 새로운 인터페이스에서 기능을 새로운 컴포넌트로 대치
		MPrmN: 새로운 인터페이스에서 새로운 기능의 매개변수의 값 변형
패턴5	*PublicSub*: "Public Sub"로 선언된 기능	*CSubS*: 새로운 시퀀스를 포함하는 새로운 기능을 생성
	PublicFunction: "Public Function"으로 선언된 기능	*MPrmS*: 새로운 시퀀스를 포함하는 기능의 매개변수의 값을 변형
패턴6	*PublicSubN*: 새로운 인터페이스에서 "Public Sub"로 선언된 기능	*CSubSN*: 새로운 인터페이스에서 새로운 시퀀스를 포함하는 기능을 생성
	PublicFunctionN: 새로운 인터페이스에서 "Public Function"으로 선언된 기능	*MPrmSN*: 새로운 인터페이스에서 새로운 시퀀스를 포함하는 기능의 매개변수의 값을 변형

4. 사례 적용

3장에서 제안한 COM 컴포넌트의 맞춤 테스트 기법을 실제 대규모 컴포넌트 기반 시스템인 샤모아에 적용한다.

4.1 샤모아

샤모아[11]는 상업용 소프트웨어와 자체 개발한 컴포넌트들을 단일한 API를 통하여 접근할 수 있도록 COM과 자바 웹 서비스로 개발한 컴포넌트 시스템이다. 샤모아는 IKEA 아래, 다음과 같은 다양한 상용 컴포넌트들과 자체개발한 컴포넌트들로 이루어져 있다.

- 상용 컴포넌트: ETL Tool(MS SQL), Data Warehouse(MS SQL, Oracle 9), API(MS OLE DB for OLAP, MS OLE DB for DM)
- 자체 개발 컴포넌트: DAQUM, OLAP Server, DBSCAN-W, COD-DBSCAN, Text Mining, Real-time multimedia delivery: Qos Routing, XML Server, XML Document Mining, Electronic-payment-system test bed

본 사례적용에서는 이 가운데 DAQUM 컴포넌트에 3장의 기법을 적용한다. DAQUM은 오류 데이타 분류(Dirty data taxonomy)[8]를 기반으로 데이타 품질을 측정하고 제어한다. 그림 2는 DAQUM을 이루는 요소들이며, 이들의 *B*와 *W*는 정의 1과 정의 2에 따라 다음

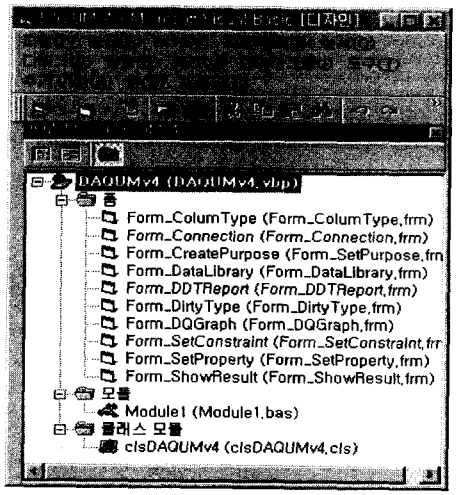


그림 2 DAQUM의 B와 W

처럼 구분되어진다.

4.2 DAQUM 맞춤

DAQUM은 COM 컴포넌트로서 표 1의 COM 컴포넌트 맞춤이 모두 가능하지만, 본 사례적용에서는 예로서 패턴 3의 맞춤을 수행한다. DSN에 성공적으로 연결되었는지를 확인하는 새로운 기능을 추가하기 위해, DAQUM을 맞춤하였다. 새로운 기능을 추가하기 위해

```
Dim Constr As String
Dim setDSN As New DSNConnection.ClsDSNCon

Public Sub DAStart()
        NewConnection
    SetProperty
    DirtyDataClean x, y
End Sub

Public Sub Connection()
    Form_Connection.Show vbModal
    Constr = Form_Connection.Constr
End Sub

Public Sub NewConnection()
  Set setDSN = new DSNConnection.ClsDSNCon
  setDSN.DSNConnection
  Constr = setDSN.Constr
End Sub
```

(a)

```
Dim Constr As String
Dim setDSN As New DSNConnection.ClsDSNCon

Public Sub DAStart()
        Connection
    SetProperty
    DirtyDataClean x, y
End Sub

Public Sub Connection()
    Form_Connection.Show vbModal
    Constr = Form_Connection.Constr
End Sub

Public Sub NewConnection()
    Set setDSN = new DSNConnection.ClsDSNCon
    setDSN.DSNConnection
    Constr = setDSN.Constr
End Sub
```

(b)

그림 3 DAQUM의 *cBW*와 *fBW*

서 그림 3(a)처럼 DSNConnection을 수정한다. 그림 3(a)는 DAQUM의 하나의 *cBW*이고, 이는 패턴 3에 해당한다. 맞춤으로 변형된 부분은 그림 3(a)에 굵은 글씨체로 표현하였다. 맞춤을 수행하는 과정에서 Constr에 예기치 않은 오류가 존재하게 되었고, 이를 3장에서 제안한 기법을 이용하여 테스트한다.

4.3 테스트 데이타 선정

테스트 데이타를 선정하기 위하여, 우선 패턴3의 *FIT*와 *FIO*를 *cBW*에 적용하여 *fBW*를 생성한다. 4.2장의 맞춤의 경우는 "Sub"로 선언된 부분의 변형으로 맞춤3을 수행하였으므로, *FIT*는 *PublicSub*와 *FIO*는 *RSub*를 사용하여, 그림 4(b)의 *fBW*를 생성하였다. *fBW*가 생성되고 나면 정의 6에 따라 테스트 데이타를 선정할 수 있다. 표 4는 정의 6에 따라 *cBW*와 *fBW*를 차별화하는 메소드 시퀀스로 선정된 테스트 데이타의 일부를 보여준다.

4.4 DAQUM 테스트

표 4의 테스트 데이타들 가운데 임의로 "DSN: Chamois, ID: sa, PWD: 1234"을 *cBW*에 적용하였다. 이에 대하여 컴포넌트 사용자는 *cBW*의 결과로서 "Form_DirtyDataClean Emp emp"를 기대하고 있으나, *cBW*에 적용한 실제 결과는 "No DSN connection value"이다. 이를 통해 *cBW*가 오류를 갖고 있음으로 감지한다.

표 4 맞춤된 DAQUM의 테스트 데이타의 일부

테스트 데이타, *x*	*cBW(x)*	*fBW(x)*
DSN: Chamois, ID: sa, PWD: 1234	No DSN connection value	Form_DirtyDataClean Emp, emp
DSN: Chamois, ID: sa, PWD:	No DSN connection value	Form_DirtyDataClean Emp, emp
DSN: Chamois, ID: , PWD: 1234	Connection failed	No DSN connection value
DSN: , ID: sa, PWD: 1234	Connection failed	No DSN connection value
DSN: Chamois, ID: , PWD:	Connection failed	No DSN connection value
...	...	...

5. 실험 분석

효율성이 좋은 테스트 데이타는 테스트 시간을 줄여줌으로써, 테스트 비용 절감에 기여할 수 있다. 이는 비용 절감을 목표 가운데 하나로 하고 있는 컴포넌트 기반 소프트웨어 개발에서는 중요한 의미를 갖는다. 따라서 본 장에서는 3장에서 제안한 본 기법이 생성하는 테스트 데이타들의 효율성을 실험을 통하여 분석한다.

5.1 효율성 척도

테스트 데이타의 효율성은 다음의 두 가지 척도, Eff_1과 Eff_2,를 가지고 측정한다.

- Eff_1 = (테스트 데이타에 의해 감지되는 오류의 수 / 전체 오류의 수) * 100
- Eff_2 = (오류를 감지하는 테스트 데이타의 수 / 전체 테스트 데이타의 수) * 100

Eff_1은 테스트 데이타가 얼마나 많은 오류를 감지하는지[9]를 측정하고, Eff_2는 테스트 데이타들 중에서 얼마나 많은 테스트 데이타가 오류를 감지하는지[10]를 측정한다.

본 실험에서는 다음의 세 가지 이유로 인터페이스 뮤테이션을 비교대상으로 한다. 첫째, 본 기법은 컴포넌트를 테스트한다. 컴포넌트를 대상으로 하는 잘 알려진 테스트 기법이 아직은 없다. 인터페이스 뮤테이션은 현재까지 알려진 기법들 가운데 본 기법이 대상으로 하고 있는 컴포넌트를 테스트 할 수 있는 기법이다. 둘째, 본

기법은 컴포넌트의 인터페이스를 테스트에서 이용한다. 인터페이스 뮤테이션은 말 그대로 컴포넌트의 인터페이스에 본 기법과 유사하게 오류를 삽입하는 방법을 사용한다. 셋째, 본 기법은 실제 컴포넌트에 적용할수 있을 만큼 구체적이다. 인터페이스 뮤테이션도 본 기법과 유사하게 실제 적용할 수 있는 수준의 연산자를 정의하고 있다. 기존의 블랙박스 테스트나 화이트 박스 테스트들이 독립 프로그램을 기준으로 하고 있는것에 반해, 인터페이스 뮤테이션을 CORBA 분산 컴포넌트를 대상으로 하고 있다. 컴포넌트 아키텍쳐와 CORBA와 같은 분산 객체 아키텍는 모두 인터페이스와 그의 구현부분으로 이루어지는 유사한 구조를 갖고 있고, 본 기법과 인터페이스 뮤테이션이 생성하는 테스트 데이타를 동일하게 일련의 메소드 시퀀스이다. 따라서 본 기법은 블랙박스 테스트나 화이트박스 테스트 기법들과의 비교보다 인터페이스 뮤테이션과의 비교하는 것이 의미가 있다.

5.2 실험 순서

인터페이스 뮤테이션과 본 기법은 모두 뮤턴트와 *fBW*를 만드는 연산자를 정의하고 있다. 본 기법의 연산자는 표 4에 기술한대로이며, 인터페이스 뮤테이션의 연산자[6]는 다음과 같다.

- *Replace*: in, out, inout중의 하나도 대치하는 연산자.
- *Swap*: 메소드 호출 부분에서 매개변수를 교환하는 연산자. 동일한 형태의 매개변수로 교환된다.
- *Twiddle*: 변수 x를 succ(x)또는 pred(x)로 대치하여, x보다 하나 큰수또는 하나 작은수로 x의 값을 변형시키는 연산자.
- *Set*: 매개변수나 반환변수에 임의의 고정된 값을 설정하는 연산자.
- *Nullify* : 객체 참조에 Null값을 지정하는 연산자.

CORBA IDL은 COM의 인터페이스와 매우 유사하다. 따라서 CORBA IDL을 기준으로 인터페이스 뮤테이션에서 정의해 놓은 위의 연산자들 모두가 COM에 적용가능하다. 본 실험에서는 이들 연산자를 사용하여 인터페이스 뮤테이션을 다음의 순서로 COM 컴포넌트에 적용한다.

인터페이스로부터 메소드들의 리스트를 작성 → 인터페이스 뮤테이션 연산자를 이용하여 뮤턴트 생성 → 뮤턴트를 가지고 테스트 데이타 선정 → 테스트 데이타를 맞춤된 COM 컴포넌트에 적용 → 테스트 데이타가 오류를 감지하는지 여부를 확인 → Eff_1과 Eff_2 측정

인터페이스 뮤테이션에 따르면, 몇 개의 연산자를 사용하여 몇 개의 뮤턴트를 생성할지는 테스트를 수행할 자원과 도메인의 능력에 따라 달라질 수 있다고 한다. 본 실험에서는 하나의 연산자를 사용하여 하나의 뮤턴트를 생성하기로 하고, 그 가운데 kill되는 뮤턴트만을 세어서, 되도록 높은 효율성이 나타날 수 있도록 했다.

본 기법은 COM 컴포넌트에 다음과 같은 순서로 적용된다.

맞춤된 COM 컴포넌트의 맞춤패턴을 분석 → 맞춤패턴의 *FIT*와 *FIO*를 가지고 *fBW*를 생성 → *fBW*와 *cBW*를 가지고 테스트 데이타르 선정 → 테스트 데이타를 맞춤된 COM 컴포넌트에 적용 → 테스트 데이타가 오류를 감지하는지를 확인 → Eff_1과 Eff_2를 측정

5.3 실험의 COM 컴포넌트들

본 실험에서는 표 5의 다섯 개의 COM 컴포넌트를 대상으로 한다. 이들은 각각의 맞춤된 컴포넌트를 갖으면 테스트 기법은 이들 맞춤된 컴포넌트에 적용된다. 효율성을 측정하기 위하며, 각 맞춤 패턴들에는 오류가 있음을 가정한다. 그 오류들은 테스트 기법을 통해 감지된다.

표 5 실험에서 사용되는 COM 컴포넌트들

COM 컴포넌트	패턴	설명
AppStack	패턴4	사용자가 입력하는 데이타를 스택을 이용하여 넣고 뺀다.
AppOperator	패턴1	사용자가 입력하는 두 개의 값과 연산자를 가지고 계산한 결과를 보여준다.
AppBooking	패턴6	식당의 테이블들을 보여주고 사용자로 하여금 원하는 좌석을 선택하게 한 후, 그 테이블에 대한 예약 번호를 발부하고 결과를 데이타베이스에 저장한다.
AppZipcode	패턴3	사용자가 입력하는 주소를 가지고 해당 우편번호를 찾아준다.
AppVoting	패턴5	사용자가 선정한 주어진 객관식 문제의 답에 대하여, 현재까지 선택된 경우의 수를 보여준다.

5.4 실험 결과 및 분석

표 5의 모든 COM 컴포넌트에 본 논문에서 제안하는 기법과 인터페이스 뮤테이션을 5.2에서 보여준 적용순서에 따라 적용하였다. 임의로 모두 20개의 테스트 데이타를 선정하도록 하였다. 표 6은 그로부터 얻은 결과 값들이다. 표 6에서 IM은 인터페이스 뮤테이션을 표시하고, FIT/FIO는 본 기법을 표시한다. 표 6의 앞 3개의 행들은 Eff_1을 측정하기 위한 결과 값들이며, 뒤 4개의 행들은 Eff_2를 측정하기 위한 결과 값들이다.

그림 4는 표 6의 값들로 계산한 Eff_1을 나타낸다. FIT/FIO의 Eff_1은 거의 100%로 나타나고 있다. FIT/FIO는 맞춤 패턴을 기반으로 *fBW*를 생성하므로, 각 *fBW*가 해당 맞춤 패턴으로 발생하는 오류에 민감하다. 이것은 맞춤 패턴은 각 패턴의 맞춤으로 변형되는 부분

표 6 실험 결과 값

	COM 컴포넌트	①	②	③	④	⑤
Eff1	오류의 수	2	2	3	3	2
	FIT/FIO가 발견한 오류의 수	2	2	3	3	2
	IM이 발견한 오류의 수	1	1	2	2	2
Eff2	*fBW*의 수	2	1	1	2	1
	뮤턴트의 수	1	4	2	2	6
	IM에서 오류를 감지한 테스트 데이타의 수	10	10	6	18	20
	FIT/FIO에서 오류를 감지한 테스트 데이타의 수	17	20	16	20	20

①AppStack, ②AppOperator, ③AppBooking, ④AppZipcode, ⑤AppVoting

과 관련이 있으며, 맞춤 오류들은 이 부분을 통해 감지되기 때문이다. 따라서 일단 맞춤 패턴에 의해 *fBW*가 생성되면, *fBW*와 *cBW*를 차별화하는 테스트 데이타는 해당 패턴의 맞춤된 부분을 통해서 발생하는 오류들을 감지할 수 있다.

그림 5에서 FIT/FIO는 IM보다 좀 더 높은 Eff_2를 갖음을 보인다. 이러한 결과를 보이는 이유는 맞춤 패턴과 *FIT*에 있다. IM은 인터페이스에 정의된 모든 메소드 시그너쳐에 모든 연산자를 적용하지만, FIT/FIO는 해당 맞춤 패턴에 따른 연산자만을 적용하여 *fBW*를 생성한다. 이렇게 생성된 FIT/FIO의 테스트 데이타는 맞춤 패턴 없이 생성된 IM의 테스트 데이타보다 더욱 맞춤 오류에 민감하다. 이는 각 맞춤 패턴의 연산자들이 해당 패턴에서 발생할 수 있는 오류들을 고려하여 만들어졌

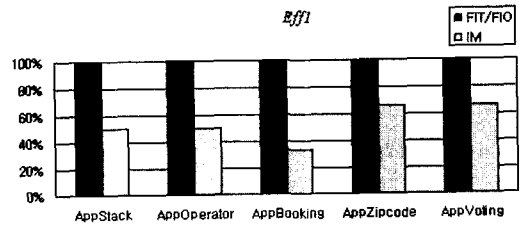


그림 4 COM 컴포넌트들의 Eff_1

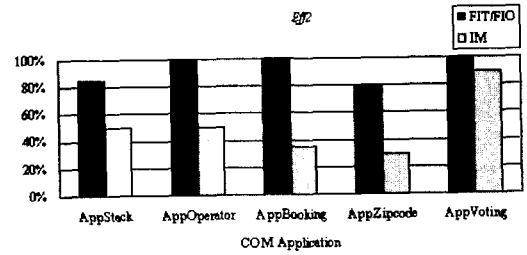


그림 5 COM 컴포넌트의 Eff_2

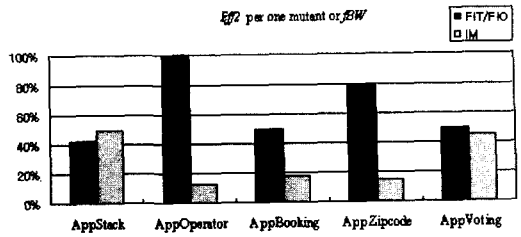


그림 6 하나의 뮤턴트 또는 *fBW*에 대한 Eff_2

기 때문이다.

만일 테스트 기법이 많은 수의 *fBW*나 뮤턴트를 필요로 한다면, 비록 높은 Eff_1이나 Eff_2를 갖더라도 테스트 비용면에서 효율적이라고 하기 힘들다. 왜냐하면, 뮤턴트나 *fBW*를 생성하고 다루는데 드는 비용이 효율성을 떨어뜨리기 때문이다. 따라서 본 실험 분석에서는 하나의 *fBW*나 뮤턴트마다 갖는 효율성을 따로 평가하였다. 이는 전체 효율성을 *fBW*의 수 또는 뮤턴트의 수로 나누어서 산출하였다. 그림 6은 그 결과를 보여주며, FIT/FIO가 그림 5에서보다 더욱 큰 차이로 IM보다 높은 효율성을 갖음을 알 수 있다.

6. 결론 및 향후 연구 과제

소스코드를 공개하지 않는 컴포넌트의 특성은 컴포넌트 사용자 입장에서의 컴포넌트 테스트를 더욱 어렵게 한다. 본 논문은 컴포넌트 맞춤 테스트 기법[3,4]을 기반으로 COM 컴포넌트를 맞춤 테스트하는 기법을 제안하였다. COM 컴포넌트는 현재 엔터프라이즈 자바빈즈와 함께 개발자들이 선호하는 컴포넌트 아키텍쳐 가운데 하나이다. 본 논문에서는 특히 비쥬얼베이직으로 작성된 COM 컴포넌트를 분석하여 맞춤 패턴과 *FIT*를 개발하였다. 또한 본 기법을 실제 대규모 컴포넌트 기반 시스템인 샤모아에 적용함으로써, 개발한 맞춤 패턴과 *FIT* 등이 실제 COM 컴포넌트에서 작동함으로 보였다. 나아가, 제안한 기법이 효율성을 갖는 테스트 데이타를 선정함으로 보이기 위하여 두 가지 수준의 효율성을 평가하는 실험을 수행하였다.

본 논문에서 제안한 기법은 개발 현장에서 많이 사용되는 COM 아키텍쳐를 기반으로 하고 있다. 컴포넌트들을 기반으로 개발된 대규모 시스템인 샤모아에서 COM 컴포넌트인 DAQUM을 추출하여 그의 맞춤 테스트를 본 기법을 이용하여 수행하였다. 이는 본 기법이 대규모 시스템에서 동작하는 COM 컴포넌트를 테스트할 수 있음을 보임으로써, 실제 컴포넌트 기반 개발 현장에 적용 가능할 정도로 구체적으로 설계된 기법임을 나타낸다.

컴포넌트 기반 개발 현장에 적용하기 위해서는 컴포넌트 기반 개발이 추구하는 목적 가운데 하나인 비용

절감에 부합되어야 한다. 이를 위하여 선정되는 테스트 데이타의 효율성을 보장하는 일이 요구된다. 본 논문에서는 효율성을 평가하기 위한 실험을 비주얼베이직 COM 컴포넌트를 대상으로 수행하였고, 그 결과 본 기법이 상대적으로 높은 효율성을 갖는 테스트 데이타를 선정함을 보였다. 이전의 논문[2,3]에서 이미 엔터프라이즈 자바빈즈를 위하여 구체화된 테스트 기법이 좋은 효율성을 지닌 테스트 데이타를 선정함으로 보였다. 이렇게 엔터프라이즈 자바빈즈뿐만 아니라 COM 컴포넌트에서도 좋은 효율성을 나타냄으로써, 본 논문에서 기반으로 하는 기법의 아이디어가 테스트 데이타의 효율성을 보장함을 알 수 있다.

엔터프라이즈 자바빈즈를 위한 맞춤 테스트 도구는 이미 개발되었다. 이는 테스트를 자동으로 수행하는 도구로서 엔터프라이느 자바빈즈를 대상으로 하는 실험에서 사용되었다[2,3]. COM을 대상으로 하는 도구도 현재 구현될 수 있으며, 나아가 본 기법의 아이디어를 현재 관심의 대상이 되고 있는 웹기반 소프트웨어 공학에 적용할 예정이다.

참 고 문 헌

[1] Johanna Ambrosio, "Testing key to Component Quality," Application Development Trends, pp.19-28, Oct. 2002.

[2] Hoijin Yoon, Byoungju Choi, "Component Customization Testing Technique Using Fault Injection Technique and Mutation Test Criteria," in Proc. Mutation 2000, San Jose, USA, pp 93-100, Oct.6-7, 2000.

[3] Hoijin Yoon, Byoungju Choi, "Effective Test Case Selection for Component Customization and Its Application to EJB," The Software Testing, Verification, and Reliability Journal, to be appeared on Vol.14 No.1 March 2004.

[4] Won Kim, Ki-Joon Chae,Dong-Sub Cho, Byoungju Choi, Anmo Jeong, Myung Kim, KiHo Lee, Meejeong Lee, Sang-Ho Lee, Seung-Soo Park, Hwan-Seung Yong, "The Chamois Component-based Knowledge Engineering Framework," IEEE Computer Journal, May 2002.

[5] Marcio E. Delamaro, Jose C. Maldonado, Aditya P. Mathur, "Integration Testing Using Interface Mutations," August 28, 1997.

[6] S. Ghosh: Testing Component-Based Distributed Applications. Ph. D Dissertation, Department of Computer Science in Purdue University. 2000.

[7] Component Object Model, http://msdn.microsoft.com/library

[8] Won Kim, Byoungju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, Doheon Lee, "A Taxonomy of Dirty Data," The Data Mining and Knowledge Discovery Journal, Vol.7, No.1. pp.81-99, 2003.

[9] W. Eric Wong, Joseph R. Horgan, Aditya P. Mathur, and Alberto Pasquini, "Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application," Proceeding of COMPSAC97. pp. 522-529, Washington D.C., USA, 1997.

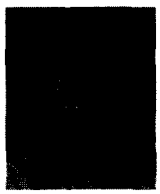
[10] Aditya P. Mathur and W. Eric Wong, "Comparing the Fault Detection Effectiveness of Mutation and Data Flow Testing: An Empirical Study," SERC-TR-146-P, Purdue University. 1993.

윤 희 진

1989년~1993년 이화여대 전산과 학사 1996년~1998년 이화여대 컴퓨터학과 석사. 1998년~2004년 이화여대 컴퓨터학과 박사. 2004년~현재 Georgia Institute of Technology College of Computing의 visiting scholar. 관심분야는 소프트웨어공학, 소프트웨어 테스트

이 병 희

1997년~2001년 세명대 소프트웨어학과 학사. 2001년~2003년 이화여자대학교 대학원 컴퓨터학과 석사. 2003년~현재 (주)키움닷컴증권 투자정보시스템팀 근무. 관심분야는 소프트웨어공학, 소프트웨어 테스트



김 은 희

1998년~2002년 경성대학교 컴퓨터공학과 학사. 2002년~2004년 이화여대 컴퓨터학과 석사. 2004년~현재 삼성전자 기술총괄 CTO 전략실. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 소프트웨어 품질 측정 및 보증

최 병 주

1979년~1983년 이화여대 수학과 학사 1984년~1985년 Purdue Univ. Computer Science 학사수료. 1986년~1987년 Purdue Univ. Computer Science 석사 1987년~1990년 Purdue Univ. Computer Science 박사. 1991년~1992년 삼성종합기술원 1992년~1995년 용인대 전산통계학과 조교수 1995년~현재 이화여대 컴퓨터학과 교수. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 소프트웨어 및 데이타 품질 측정